Machine Learning Basics - Set 3b:
Implementing a Neural Network with JAX

Datasets: A, B from previous sets and C: a tricker dataset with two features and points belonging either to class 0 or 1.

1. **Setting up the Neural Network:**

   a) Implement the following functions to initialize network parameters. Remember to use JAX's random number generation for reproducibility.

   ```
   def random_layer_params(m, n, key, scale=1e-1):
     # a layer is initialized as a set of random weights and biases
     w_key, b_key = random.split(key)
     return scale * random.normal(w_key, (n, m)), scale * random.normal(b_key, (n,))

   def init_network_params(sizes, key):
     # Your implementation here
   ```

   b) Initialize parameters for a network with layer sizes [input_dim, 64, 64, 1]. Think about what the input dimension should be based on the dataset.

   c) Implement the ReLU activation function for inner layers and the Sigmoid activation function for the last layer.

2. **Implementing the Prediction Function:**

   a) Implement the forward pass for a single example: This will involve matrix multiplication and applying the activation function.

   ```
   def predict(params, data):
       # Your implementation here
   ```

   b) Use `jax.vmap` to create a batched version of `predict`. This allows you to efficiently process multiple data points simultaneously without writing explicit loops.

   c) Test your batched predict function on dummy data. This is a good way to catch errors early on.

3. **Loss and Accuracy Functions:**

   a) Implement the loss function, in this case a binary cross-entropy: Cross-entropy loss increases as the predicted probability diverges from the actual label.

   ```
   def loss(params, data, targets):
       # Your implementation here
   ```

   b) Implement the accuracy function: This function calculates the percentage of correctly classified examples. You may also want to write a function for computing the signal and background efficiencies.

   ```
   def accuracy(params, data, targets):
       # Your implementation here
   ```

4. **Training Loop:**

   a) Implement the update function using `jax.grad` to automatically compute gradients and `jax.jit` for just-in-time compilation and speed improvements.

```
@jit
def update(params, x, y):
    grads = grad(loss)(params, x, y)
    # Your implementation here
```

b) Implement a simple training loop that iterates over batches of data. In each iteration, you'll need to update the parameters (by calculating the loss) and track the accuracy.

c) Train the network on datasets A, B, and C, reporting accuracy after each epoch. An epoch is one complete pass through the entire training dataset. Compare the performance across the datasets and with the methods of previous exercises.

Note: Use JAX's numpy (`jax.numpy as jnp`) for all numerical operations.